Self-Clustering Recurrent Networks

Zheng Zeng, Rodney M. Goodman
Department of Electrical Engineering, 116-81
California Institute of Technology
Pasadena, CA 91125

Padhraic Smyth
Jet Propulsion Laboratory, 238-420
California Institute of Technology
Pasadena, CA 91109

***Abstract*—Recurrent neural networks have recently been shown to have the ability to learn finite state automata (FSA's) from examples. In this paper it is shown, based on empirical analyses, that second-order networks which are trained to learn FSA's tend to form discrete clusters as the state representation in the hidden unit activation space. This observation is used to define 'self-clustering' networks which automatically extract discrete state machines from the learned network. However, the problem of instability on long test strings is a factor in the generalization performance of recurrent networks — in essence, because of the analog nature of the state representation, the network gradually "forgets" where the individual state regions are. To address this problem a new network structure is introduced whereby the network uses quantization in the feedback path to force the learning of discrete states. Experimental results show that the new method learns FSA's just as well as existing methods in the literature but with the significant advantage of being stable on test strings of arbitrary length.**

I. Introduction

VARIOUS direct search algorithms have been proposed for learning grammars from positive and negative examples (strings) [1,4,8,11]. More recently recurrent neural networks have been investigated as an alternative method for learning simple grammars [2,3,5,7,9]. All of these methods have shown the capability of recurrent networks to learn different types of simple grammars from examples.

In this paper we restrict the focus to a recurrent network's behavior in learning regular grammars — regular grammars are the simplest type of grammar in the Chomsky hierarchy and have a one-to-one correspondence to finite state machines [6].

The motivation of this work is to obtain a clearer understanding of the behavior of recurrent networks, in particular with regard to both their learning characteristics and internal state representations. In turn, this information may give more insight into their capability for fulfilling other more complicated tasks.

Giles et al. have proposed a "2nd-order" recurrent network structure to learn regular languages [5]. Henceforth, all references to 2nd-order recurrent networks imply the network structure described in [5]. The experiments reported in this paper confirm their claim that 2nd-order nets can learn various grammars well. In addition, we have found empirically that this structure learns these grammars more easily than the simple recurrent network structure (or the Elman structure) [3] which does not use 2nd-order units. However, a stability problem emerges with the 2nd-order trained networks as longer and longer input strings are presented (similar behavior in recurrent networks has been found in different contexts [9,10]). The stability problem led us to look deeper into the internal representation of states in such a network and the following interesting behavior was observed: during learning, the network attempts to form clusters in hidden unit space as its representation of states. This behavior occurred in *all* the learning experiments we performed. Once formed, the clusters are stable for short strings, i.e., strings with lengths not much longer than the maximum length of training strings. However, in 14 out of 15 learned networks,when sufficiently long strings are presented for testing, the clusters (states) start to merge and ultimately become indistinguishable. To solve this problem, in this paper we propose a discretized combined network structure which can be shown to successfully learn stable state representations. In the new network, instead of regions of activation space, the states of the network are actually isolated discrete points in the hidden unit activation

The research described in this paper was supported in part by DARPA under grants number AFOSR–90–0199 and NOOO14-92-J-1860. In addition this work was carried out in part by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

0-7803-0999-5/93/$03.00 ©1993 IEEE

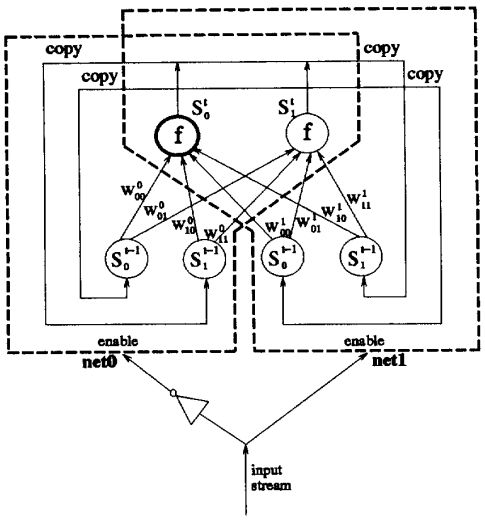

Fig. 1. Equivalent first-order structure of second-order network

space.

II. An Equivalent Gating Model For Second-Order Networks

A convenient way to view 2nd-order networks which deal with binary sequences is to decompose the network structure into two separate component networks controlled by an "enabling" or "gating" signal (Fig. 1). The network consists of two 1st order networks with shared hidden units. The common hidden unit values are copied back to both **net0** and **net1** after each time step, and the input stream acts as a switching control to enable or disable one of the two nets. For example, when the current input is 0, **net0** is enabled while **net1** is disabled. The hidden unit values are then decided by the hidden unit values from the previous time step weighted by the weights in **net0**. The hidden unit activation function is the standard sigmoid function, $f(x) = \frac{1}{1+e^{-x}}$. Note that this representation of a 2nd-order network, as two networks with a gating function, provides insight into the nature of 2nd order nets, i.e., clearly they have greater representational power than a single simple recurrent network, given the same number of hidden units.

III. The Learning Procedure And The Extraction Of State Machines

We use S_i^t to denote the activation value of hidden unit number i at time step t. w_{ij}^n is the weight from layer 1 node j to layer 2 node i in **net**n. $n = 0$ or 1 in the case of binary inputs. Hidden node S_0^t is chosen to be a special indicator node, whose activation should be close to 1 at the end of a legal string, or close to 0 otherwise.

A standard 2nd-order network (or equivalently, the gated network of Fig. 1) was used in all of the experiments described in this section. The following grammars were used in our initial experiments:

- Tomita grammars [11]:
  - #1 — 1^*
  - #4 — any string not containing "000" as a substring.
  - #5 — $[(01|10)(01|10)]^*$
  - #7 — $0^*1^*0^*1^*$
- Simple vending machine: The machine takes in 3 types of coins: nickel, dime and quarter. Starting from empty, a string of coins is entered into the machine. The machine "accepts", i.e., a candy bar may be selected, only if the total amount of money entered exceeds 30 cents.

A training set consists of randomly chosen variable length strings with length uniformly distributed between 1 and L_{max}, where L_{max} is the maximum training string length. Each string is marked as "legal" or "illegal" according to the underlying grammar. The learning procedure is a gradient descent method in weight space (similar to that proposed by Williams and Zipser [12]) to minimize the error at the indicator node for each training string [5].

In a manner different from that described in [5], we present the whole training set (which consists of 100 to 300 strings with L_{max} in the range of 10 to 20), all at once to the network for learning, instead of presenting a portion of it in the beginning and gradually augmenting it as training proceeds. Also, we did not add any *end* symbol to the alphabet as in [5]. We found that the network can successfully learn the machines (2–7 states) we tested on, with a small number of hidden units (4–5) and with training in less than 500 epochs. This agrees with the results described in [5].

To examine how the network forms its internal representation of states, we recorded the hidden unit activations at every time step of every training string in different training epochs. As a typical example, shown in Fig. 2(a)-(e) are the $S_0 - S_3$ activation-space records of the learning process of a 4-hidden-unit network. The underlying grammar was Tomita #4, and the training set consisted of 100 random strings with $L_{max} = 15$. Note that here the dimension S_0 is chosen because of it being the important "indicator node", and S_3 is chosen arbitrarily. The observations that follow can be made from any of such 2-D plots from any

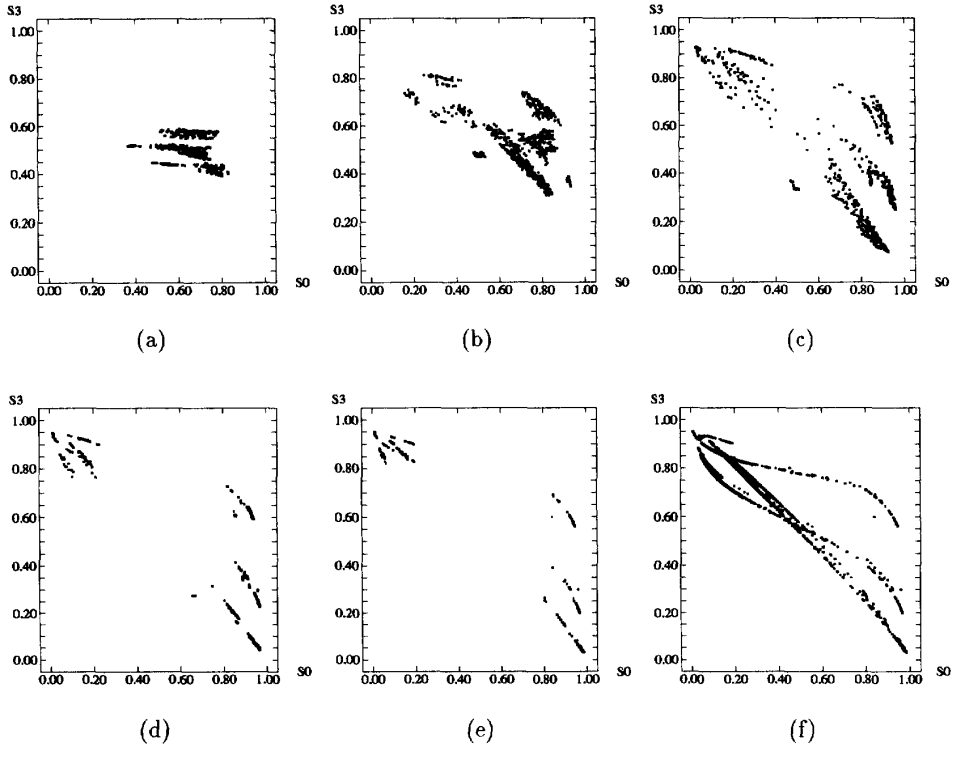

Fig. 2. Hidden unit activation plot $S_0 - S_3$ in learning Tomita grammar #4. (S_0 is the x axis.) (a)-(e) are plots of all activations on the training data set. (a) During 1st epoch of training. (b) During 16th epoch of training. (c) During 21st epoch of training. (d) During 31st epoch of training. (e) After 52 epochs, training succeeds, weights are fixed. (f) After training, when tested on a set of maximum length 50.

run in learning any of the grammars in the experiments. Each point corresponds to the activation pattern of a certain time step in a certain string. Each plot contains the activation points of *all* time steps for *all* training strings in a certain training epoch as described in the caption.

It can be observed that the network attempts to form clusters in activation space as its own representation of states and is successful in doing so. When given a string, the activation point of the network jumps from cluster to cluster as input bits are read in one by one. Hence, the behavior of the network looks just like a state machine's behavior. Motivated by these observations, we applied the k-means clustering algorithm to the activation record in activation space of the trained network to extract the states (rather than dividing up the space evenly as in [5]). Empirically, results were more reliable when k was chosen to be a large number, for example, 20. The initial seeds were chosen randomly. We then defined each cluster found by the k-means algorithm to be a "state" of the network and used the center of each cluster as a representative of the state. The transition rules for the resulting state machine were calculated by setting the S_i^{t-1} nodes equal to a cluster center, then applying an input bit (0 or 1 in binary alphabet case), and calculating the value of the S_i^t nodes. The transition from the current state given the input bit is to the state that has a center closest in Euclidean distance to the obtained S_i^t values. We then applied Moore's state machine reduction algorithm on the originally extracted machine to get an equivalent minimal machine which accepts the same language but with the fewest possible number of states. In this manner we were able to extract machines that are equivalent to the minimal machines corresponding to the underlying grammars from which the data was generated.

IV. The Stability Problem With Long Test Strings

With the 2nd-order networks described above, the trained networks perform well in classifying unseen short strings (not much longer than L_{max}), but as longer and longer strings are presented to the network, the percentage of strings correctly classified drops substantially. Shown in

Fig. 2(f) is the recorded activation points for S_0-S_3 of the same trained network from Fig. 2(e) when long strings are presented. The original net was trained on 100 strings with $L_{max} = 15$, whereas the maximum length of the test strings in Fig. 2(e) was 50. Activation points at all time steps for all test strings are shown.

It can be seen that the well-separated clusters formed during training begin to merge together for longer and longer strings and eventually become indistinguishable. These points in the center of Fig. 2(e) correspond to activations at time steps longer than $L_{max} = 15$, which means that network could not make hard decisions on long strings. The activation points of a string stay in the original clusters for short strings and start to diverge from them when strings become longer and longer. The diverging trajectories of the points form curves with sigmoidal shape.

Similar behavior was observed for 14 out of 15 of the networks successfully trained on different machines, excluding the vending machine model which is somewhat of a pathological case since all long strings are legal.

V. A Self-Clustering Network That Can Form Stable States

From the above experiments it is clear that even though the network is successful in forming clusters as its state representation during training, it has difficulty in creating *stable* clusters, i.e., to form clusters such that the activation points for long strings converge to certain centers of each cluster, instead of diverging as observed in our experiments. The problem can be considered as inherent to the structure of the network where it uses analog values to represent states, while the states in the underlying state machine are actually discrete. One intuitive suggestion to fix the problem is to replace the analog sigmoid activation function in the hidden units with a threshold function:

$$D(x) = \begin{cases} 1.0 & \text{if } x \geq 0.5 \\ 0.0 & \text{if } x < 0.5 \end{cases}$$

In this manner, once the network is trained, its representation of states (i.e., activation pattern of hidden units) will be stable and the activation points won't diverge from these state representations once they are formed. However, there is no known method to train such a network, since one can not take gradient of such activation functions.

An alternative approach would be to train the original 2nd-order network as described earlier, but to add the discretization function $D(x)$ on the copy back links during testing. The problem with this method is that one does not know *a priori* where the formed clusters from training will be. Hence, one does not have good discretization

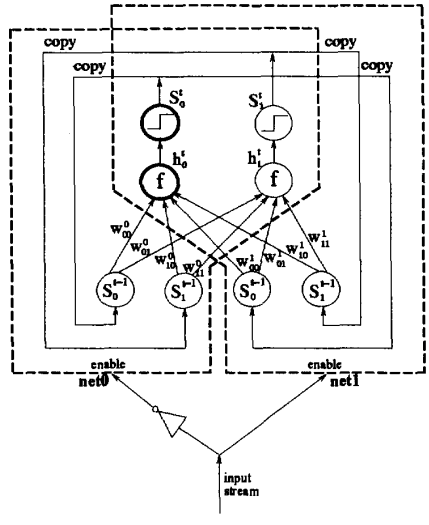

Fig. 3. A combined network with discretization

values to threshold the analog values in order for the discretized activations to be reset to a cluster center. Experimental results have confirmed this prediction. For example, after adding the discretization, the modified network can't even correctly classify the training set which it has successfully learned in training. As in the previous example, after training and without the discretization, the network's classification rate on the training set was 100%, while with the discretization added, the rate became 85%. For test sets of longer strings, the rates with discretization were even worse.

We propose that the discretization be included in *both* training and testing in the following manner: Fig. 3 shows the structure of the network with discretization added.

From the formulae below, one can clearly see that in operational mode, i.e. when *testing*, the network is equivalent to a network with discretization only:

$$\begin{aligned} h_i^t &= f(\sum_j w_{ij}^{x^t} S_j^{t-1}), \qquad \forall i, t \\ S_i^t &= D(h_i^t), \\ &\quad \textit{where } D(x) = \begin{cases} 0.8 & \text{if } x \geq 0.5 \\ 0.2 & \text{if } x < 0.5 \end{cases} \\ \Rightarrow S_i^t &= D(\, f(\sum_j w_{ij}^{x^t} S_j^{t-1})\,) \\ &\equiv D_0(\sum_j w_{ij}^{x^t} S_j^{t-1}), \end{aligned}$$

$$where \quad D_0(x) = \begin{cases} 0.8 & \text{if } x \geq 0.0 \\ 0.2 & \text{if } x < 0.0 \end{cases}$$

(Here x^t is the input bit at time step t. We use h_i^t to denote the analog value of hidden unit i at time step t, and S_i^t the discretized value of hidden unit i at time step t.)

The sigmoid nodes can be eliminated in testing to simplify computation.

During training, however, the gradient of the soft sigmoid function is made use of as a "pseudo gradient" [13] to provide a heuristic hint as to which direction (and how close) a step up or a step down would be in the thresholded output function.

By adding these discretizations into the network, one might argue that the capacity of the net is greatly reduced, since each node can now take on only 2 distinct values, as opposed to infinitely many values (at least in theory) in the case of the undiscretized networks. However, in the case of learning discrete state machines, the argument depends on the definition of the capacity of the analog network. In our experiments, 14 out of 15 of the learned networks have unstable behavior for nontrivial long strings, so one can say that the capability of such networks to distinguish different states may start high, but deteriorates over time, and would eventually become zero.

VI. Experimental Results

Shown in Fig. 4(a),(b),(c) are the $h_0 - h_1$ activation-space records of the learning process of a discretized network (h values are the undiscretized values from the sigmoids). The grammar being learned is again the Tomita grammar #4. The parameters of the network and the training set are the same as in the previous case. Again, any of the other 2-D plots from any run in learning any of the grammars in the experiments could have been used here.

Fig. 4(c) is the final result after learning, where the weights are fixed. Notice that there are only a finite number of points in the final plot in the analog activation h-space due to the discretization. Fig. 4(d) shows the discretized value plot in $S_0 - S_1$, where only 3 points can be seen. Each point in the discretized activation S-space is automatically defined as a state. The transition rules are calculated as before, and an internal state machine in the network is thus constructed. In this manner, the network performs self-clustering. For this example, 6 points were found in S-space, so a 6-state-machine was constructed as shown in Fig. 5(a). Not surprisingly this machine reduces by Moore's algorithm to a minimum machine with 4 states which is exactly the Tomita grammar #4 (Fig. 5(b)). Similar results were observed for all the other grammars in the experiments.

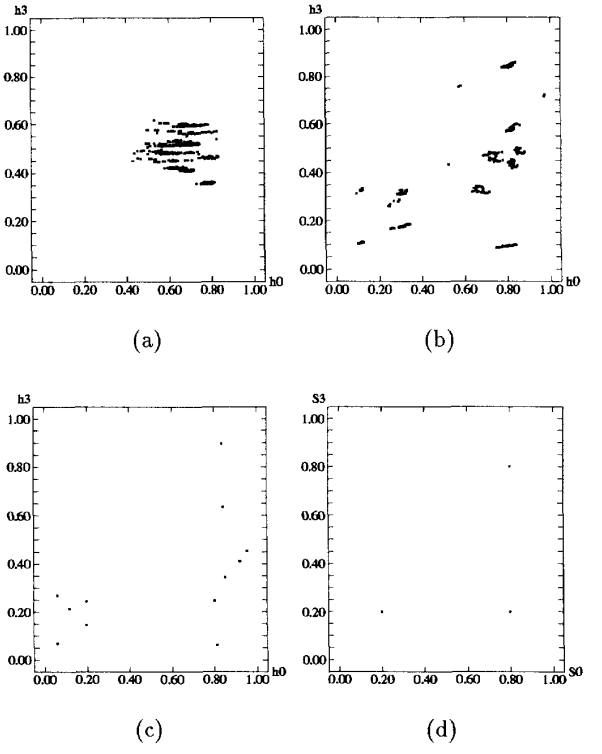

Fig. 4. Discretized network learning Tomita grammar #4. (a) $h_0 - h_3$ during 1st epoch of training. (b) $h_0 - h_3$ during 15th epoch of training. (c) $h_0 - h_3$ after 27 epochs when training succeeds, weights are fixed. (d) $S_0 - S_3$, the discretized copy of $h_0 - h_3$ in (c).

There are several advantages from introducing discretization into the network:

1. Once the network has successfully learned the state machine from the training set, it's internal states are stable. It will always classify input strings correctly, independent of the length of these strings.

2. No clustering is needed to extract out the underlying state machine, since instead of using vague clusters as its states, the network has formed distinct, isolated points as states. Each point in hidden-unit activation space is a state. The network behaves *exactly* like a state machine with no instability whatsoever.

3. Experimental results show that the size of the state machines extracted out in this approach are of a much smaller size than that found previously by the clustering method (Section III). Note that in the new discretized structure there is no need to man-

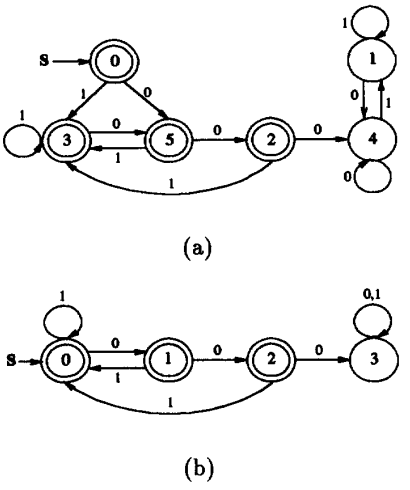

Fig. 5. Extracted state machine from the discretized network after learning Tomita grammar #4. (double circle means "accept" state, single circle means "reject" state.) (a) 6-state machine extracted directly from the discrete activation space. (b) Equivalent minimal machine of (a).

ually choose the number of clusters to "discover" the internal network states (no need to choose k for k-means) — in this sense the network performs self-clustering.

It should be noted that convergence has a different meaning in the case of training discrete networks as opposed to the case of training analog networks. In the analog networks' case, learning is considered to have converged when the error for each sample is below a certain *error tolerance level*. In the case of discrete networks, however, learning is only stopped and considered to have converged when *zero error* is obtained on all samples in the training set. In the experiments reported in this paper the analog tolerance level was set to 0.2. The discretized networks took on average 30% longer to train in terms of learning epochs compared to the analog networks for this specific error tolerance level.

VII. Conclusions

In this paper we explored the formation of clusters in hidden unit activation space as an internal state representation for 2nd-order recurrent networks which learn regular grammars.

These states formed by such a network during learning are not a stable representation, i.e., when long strings are seen by the network the states merge into each other and eventually become indistinguishable.

We introduced a new network structure which uses hard-limiting threshold discretization in the feedback path. Experimental results show that the network has similar capabilities in learning finite state machines as the original 2nd-order network, but is stable regardless of string length since the internal representation of states in this network consists of isolated points in activation space.

References

[1] D. Angluin, C. H. Smith, "Inductive inference: theory and methods," *ACM Computing Survey*, Vol.15, No.3, p.237, 1983.

[2] A. Cleeremans, D. Servan-Schreiber, J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural Computation*, Vol.1, pp.372-381, 1989.

[3] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine Learning*, Vol.7, No.s 2/3, pp.195-225, 1991.

[4] K. S. Fu, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[5] C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, Y. C. Lee, "Second-order recurrent neural networks," *Neural Computation*, Vol.4, No.3, pp.393-409, 1992.

[6] J. E. Hopfcroft, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading Mass., 1979.

[7] M. I. Jordan, *Serial Order: A Parallel Distributed Processing Approach*, Tech. Rep. No.8604, San Diego: University of California, Institute for Cognitive Science, 1986

[8] S. Muggleton, *Grammatical Induction Theory*, Addison-Wesley, Turing Institute Press, 1990.

[9] J. B. Pollack, "The induction of dynamical recognizers," *Machine Learning*, Vol.7, No.s 2/3, pp.227-252, 1991.

[10] D. Servan-Schreiber, A. Cleeremans, J. L. McClelland, "Graded state machines: the representation of temporal contingencies in simple recurrent networks," *Machine Learning*, Vol.7, No.s 2/3, pp.161-193, 1991.

[11] M. Tomita, "Dynamic construction of finite-state automata from examples using hill-climbing," *Proceedings of the Fourth Annual Cognitive Science Conference*, pp.105, 1982.

[12] R. J. Williams, D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, Vol.1, No.2, pp.270-280, 1989.

[13] Z. Zeng, R. Goodman, P. Smyth, "Learning finite state machines with self-clustering recurrent networks," *Neural Computation*, in press.